KRYPTONO PAYMENT SDK INSTRUCTIONS

❑ Accounts Setup

In order to use our payment SDK system, you will need to have **3 different accounts** on Kryptono Exchange to manage your funds, and monitor your selling and trading activities. Each account will allow you to perform different actions as follows:

(1) Funding Account: To receive payment from your users when they transfer funds on Kryptono Exchange into your account to purchase your native token.

(2) Selling Account: To receive payment from your users when they exchange other crypto with your native token, or buy your native token with credit card.

(3) Merchant Account: To manage your Funding Account and Selling Account.

❑ How to Set Up your Accounts

Go to kryptono.exchange to create both the Funding Account and Selling Account. Please note that the registration for these 2 accounts requires **corporate documents** in order to be successfully processed. Refer to our Enterprise Verification Guidelines for more information.

Once set up, the way that funds will flow between your Funding Account and Selling Account are so organized as to allow for better tracking of any potential discrepancies, and hence is more precise accounting for you.

1. Funding Account

The Funding Account is the acquisition of users' payments for your native token collected by Kryptono Exchange. Users can choose to top up tokens into their account on your application by clicking "Transfer" on SDK application. This amount will then be transferred into your Funding Account and as the final step of this process, to their corresponding account on your application.

For this transaction type, **5% of a user's deposited tokens** will automatically be deducted from the user's payment as a payment fee borne by you. This fee is incurred upon each successful payment made by a user. (For example, if a user transfers 100 SPIKE, our system will automatically deduct 5% or 5 SPIKE, and you receive 95 SPIKE in payment.)

The Funding Account can also facilitate the distribution of users' tokens on Kryptono Exchange into their accounts on your application. It acts as a support to your Selling Account whereby the

accumulated amount of tokens received in the Selling Account will be transferred into the Funding Account to be distributed to users on your app at a later time.

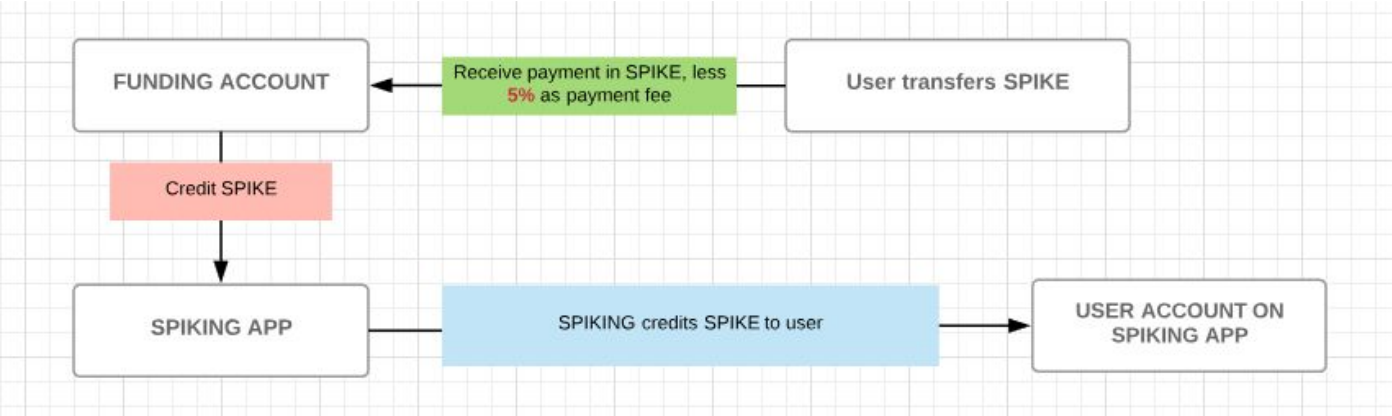


Figure 1. How the Funding Account Works

2. Selling Account

This account is where any exchange related activity takes place. In other words, users can exchange crypto or credit card for your native tokens available in the Selling Account. Thus, it is important for you to have sufficient balance for various coins in this account at all times (see our guide on how to deposit your crypto).

There are **two** ways a user can purchase your token, as follows:

Purchase with Crypto: When a user exchanges other crypto for your native token, this amount of crypto will be transferred into your Selling Account. At the same time, the Selling Account will extract the requisite amount of tokens and place them in the Funding Account (exchange rate is subject to market changes). You can then transfer the tokens from your Funding Account to the the user's account on your application.

For this transaction type, **0.1% of a user's crypto** will automatically be deducted from the user's payment as a payment fee borne by you. This fee is incurred upon each successful payment made by a user.

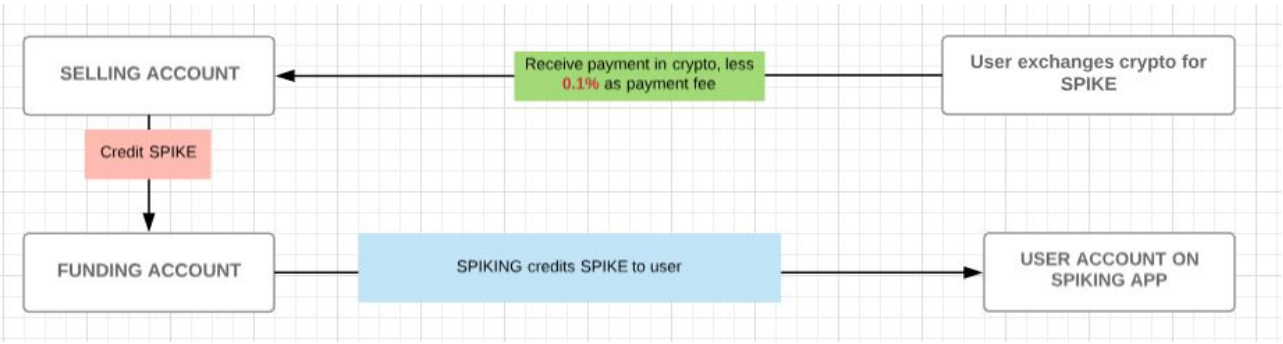


Figure 2.1. How the Selling Account Works (Crypto)

Purchase with Credit Card: When a user purchases your native token by credit card, this amount in USD will be transferred into your Selling Account. At the same time, the Selling Account will extract the requisite amount of tokens and place them in the Funding Account (USDT exchange rate is subject to market changes). You can then transfer the tokens from your Funding Account to the user's account on your application.

With this payment method, users are not required to log in to Kryptono Exchange. However, there will be a **minimum and maximum purchase limit** imposed on each transaction, which are USD10 and USD250 respectively.

For this transaction type, **5% of a user's USD** will automatically be deducted from the user's payment as a payment fee borne by you. This fee is incurred upon each successful payment made by a user.

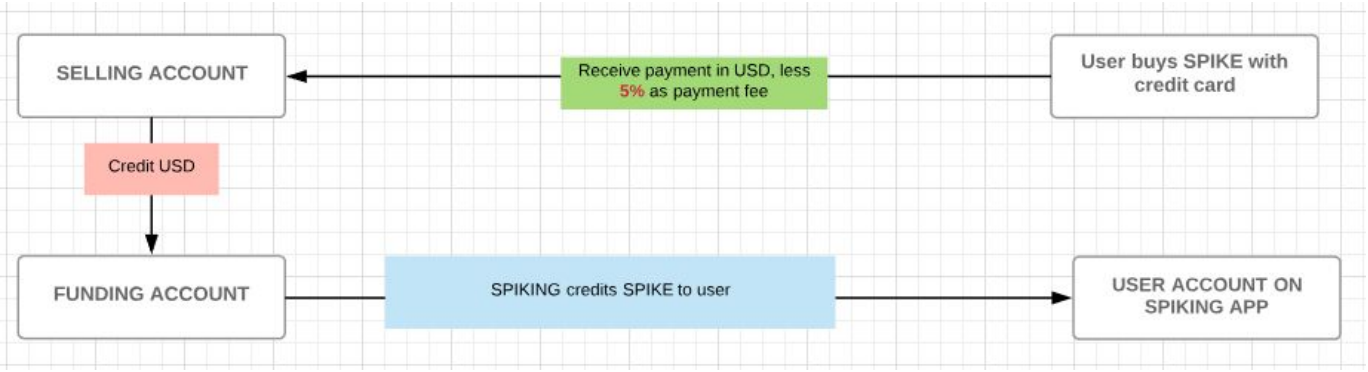


Figure 2.2 How the Selling Account Works (Credit Card)

3. Merchant Account

This is your master account which is used to manage Funding Account and Selling Account. General information related to administration activities can be tracked here, including your Balance,

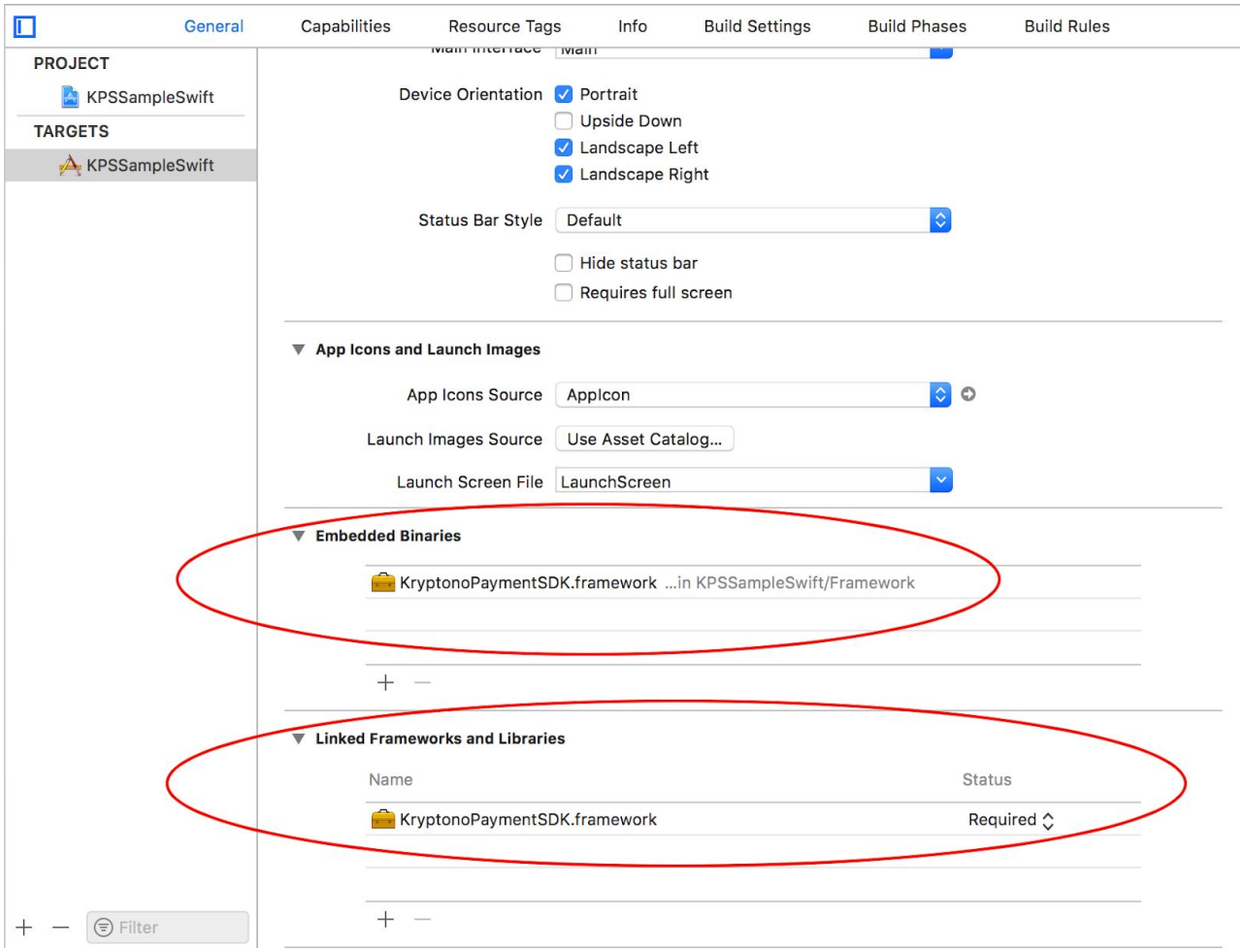
Transaction History, and your Merchant API Key. **Kryptono will generate this account for you**, and provide you with the details of this account via email.

❑ **Technical Integration Guide on Client Side**

I. iOS Platform (Version 1.0)

1. How to Integrate Kryptono Payment SDK on your Project

- Drag file KryptonoPaymentSDK.framework to the project.
- Select tab General then add KryptonoPaymentSDK on section Embedded Binaries and Linked Frameworks and Libraries.



- If your project is written in Objective C, select tab Build Settings, set value to YES for key 'Always Embed Swift Standard Libraries'.

2. How to Use Kryptono Payment SDK

- **Step 1**: Import following module.

+ If your project is written in Swift

```
import KryptonoPaymentSDK
```

+ If your project is written in Objective C

```
#import <KryptonoPaymentSDK/KryptonoPaymentSDK.h>
```

- **Step 2**: Get Client Key, Checksum Key (available on Admin site).

- **Step 3**: MUST call the following class method before calling any other methods on the framework. Otherwise, it will cause failure.

```
+ (BOOL)configureWithClientKey:(NSString *)clientKey
                   checksumKey:(NSString *)checksumKey;
```

+ Parameters:
  . clientKey: Go to Admin site to get your client key
  . checksumKey: Go to Admin site to get your checksum key

- **Step 4**: (Optional) By default, SDK will be in English. If your application wants to change the language of the KryptonoPaymentSDK, just call this method.

```
+ (KPFError)setDesiredLanguage:(KPFSupportedLocalization)lang;
```

+ Parameters:
  . lang: supported languages. Check enum KPFSupportedLocalization for more detail

- **Step 5**: Launch SDK by calling following one of following class method.

OPTION 1: ALLOW PASSING FIXED AMOUNT TO SDK (FOR PAYMENT PURPOSE)
  => SDK will take exactly the fixed amount and return to your app

```
+ (void)launchFromController:(UIViewController * _Nonnull )sourceController
                  userIdentity:(NSString * _Nonnull )userId
                      delegate:(id<KPFrameworkProtocol>)delegate
    yourCryptoRequestedAmount:(double)amount;
```

+ Parameters:

. sourceController: controller that is used to launch sdk

. delegate: an instance implementing protocol KPFrameworkProtocol to receive the payment result.

. userIdentity: It's accountId or userId or any name that your platform uses to identify the user.

. amount: fixed amount (in YOUR CRYPTO SYMBOL) passed to SDK. It must be greater than zero.

+ Discussion: error code 'KPFError_Requested_Amount_Must_Be_Greater_Than_Zero' may be returned if 'amount' argument less or equal to zero

OPTION 2: NOT ALLOW PASSING AMOUNT TO SDK (FOR TOPUP PURPOSE)

=> Users can enter an amount they want to top up

```
+ (void)launchFromController:( UIViewController * _Nonnull )sourceController
                  userIdentity:( NSString * _Nonnull )userId
                      delegate:(id<KPFrameworkProtocol>)delegate;
```

+ Parameters:

. sourceController: controller that is used to launch sdk

. delegate: an instance implementing protocol KPFrameworkProtocol to received the payment result.

. userIdentity: It's accountId or userId or any name that your platform uses to identify the user.

- **Step 6**: Implement KPFrameworkProtocol to receive payment result.

```
@protocol KPFrameworkProtocol <NSObject>

@required
- (void)kpFramework:(KPFramework *)kpframework
  didReceivePayment:(NSString *)paymentId
              error:(KPFError)error;

@end
```

*** **Note**:

- Check KPFError to get all errors returned from SDK.

```
enum kpf_error {
    KPFError_No_Error = 0,

    //Configuration
    KPFError_Framework_Not_Configured,

    KPFError_InvalidClientKey,
    KPFError_InvalidChecksumKey,
    KPFError_EnterpriseNotFound,
    KPFError_WrongClientKeyType,

    //User
    kPFError_User_Identity_Is_Null,

    //Language
    KPFError_Language_Not_Supported,

    //
    KPFError_Requested_Amount_Must_Be_Greater_Than_Zero
};
typedef enum kpf_error KPFError;
```

- Check KPFSupportedLocalization to get all supported languages

```
enum kpf_supported_localization {
    kPFSL_English = 0,
    kPFSL_ChineseSimplified
};
typedef enum kpf_supported_localization KPFSupportedLocalization;
```

II. HTML Platform

How to Integrate Kryptono Payment SDK to your project

1. Installing the SDK

- Load this script into your (local/testing) environment

```
  <script
src="https://payment.kryptono.exchange/static/script.test.js"></script>
```

- For production, please switch to this script

```
  <script
src="https://payment.kryptono.exchange/static/script.js"></script>
```

2. Init SDK

- Call init method to init client key and checksum key after loaded script
- Without init client key and checksumKey , SDK can not working correctly
- Init Localization with lang field, default is English, supported language:
  - en - English
  - zh - Chinese Traditional

```
<script>
  KryptonoPaymentSDK.init({
    clientKey:
'eyJhbGciOiJIUzUxMiJ9.eyJjcmVhdGVkQXQiOjE1MzI0MDI3NjQ4MTgsInN1YiI6IiIsImVud
GVycHJpc2VJRCI6IjRlMjU3YjZkLTE3NGEtNDcxMC04NWYxLWUzMjlkMGUwNjBlMyIsInR5cGUi
OiJjbGllbnRLZXkiLCJleHAiOjQxMTg1NzY0MDAsImlhdCI6MTUzMjQwMjc2NH0.imm8hYVMRI4
NvPnKOssYGGZy2INkfGPky93hPO_vuwl6LS3nEAgXF_gxbCM1JzFB6QzsRu47KawEu0uvjtjV0A
',
    checksumKey: 'VmCwoHo+Zx/rLs0tixQmqizz8hwft5JR3MDkiXBqq2g=',
    lang: 'en',
  })
</script>
```

- *For production please switch your client key and checksum key into production key*

How to Request a Transfer/Payment

OPTION 1: FOR PAYMENT PURPOSE (PASSING FIXED AMOUNT TO SDK)
=> SDK will take exactly the fixed amount and return to your app

Method

`KryptonoPaymentSDK.payment`

Params

Field	Descriptions	Type	Required
`userIdGetter`	callback hook for getting user identity, required for identity user.	`callback() :userId|Promise<userId>`	required
`amount`	Amount of requested base currency.	`number`	required

Return

- Promise base PaymentId

Errors

```
{
  error: "400501",
  error_description: "Invalid client Key"
}
```

Error Code	Error Description
400901	User closed Window
400902	Top up/Payment window already opened
400903	userIdGetter is missing or not a callback
400904	invalid or missing Amount
400501	Invalid client Key

400503	Invalid checksum key
404501	Enterprise not found
406501	Wrong client key type

Example

```
KryptonoPaymentSDK.payment({
  userIdGetter: function(){
    /**
     * Return your userId here
     * Or returned Promise base <userId>
     * */
    return userId
  },
  /***/
  amount : 3000
})
  .then(paymentId => {
    console.log('Payment ID returned %s', paymentId);
  })
  .catch(error => {
    if(error)
      alert(error.error_description)
  })
```

Live example can found here: https://codepen.io/kryptono/pen/djgjoM

OPTION 2: FOR TOPUP PURPOSE (USER TO INPUT AMOUNT)

=> Users can enter an amount they want to top up

Method

`KryptonoPaymentSDK.launch`

Params

Field	Descriptions	Type	Required
`onPaymentId`	callback when user confirmed an payment	`callback(id): void`	optional

`userIdGetter`	callback hook for getting user identity, required for identity user.	`callback() :userId|Promise<userId>`	required

Return

- Promise base List Payment

Errors

```
{
  error: "400501",
  error_description: "Invalid client Key"
}
```

Error Code	Error Description
400901	User closed Window
400902	Top up/Payment window already opened
400903	userIdGetter is missing or not a callback
400501	Invalid client Key
400503	Invalid checksum key
404501	Enterprise not found
406501	Wrong client key type

Example

```
KryptonoPaymentSDK.launch({
  onPaymentId: function(id){
    /**
     * This callback will be fired after user confirmed an payment.
     * */
    console.log('on new payment returned %s', id);
```

```
  },
  userIdGetter: function(){
    /**
     * Return your userId here
     * Or returned Promise base <userId>
     * */
    return userId
  }
})
  .then(listPaymentId => {
    /**
     * listPaymentId returned after user pay and close Topup window
    */
    console.log('Transaction ID List returned');
    for(var i in listPaymentId)
      console.log(listPaymentId[i])
  })
  .catch(error => {
    /**
     * Processing Error Here
    */
    if(error)
      alert(error.error_description)
  })
```

Live example can found here: https://codepen.io/kryptono/pen/qyyJde

❑ Technical Integration Guides on Server side

I. HMAC SHA256 Signature

1. Introduction

- Hmac SHA256 is used to verify data checksum
- Checksum hash is created from the request body as string and a checksum key using Hmac SHA256 algorithm.
- See example for Hmac SHA256 here.

2. Example

- Here is a step-by-step example of how to generate a valid signed payload from the Linux command line using echo, openssl.
    - Request Body

```
{"paymentId":"c0dcef90-6caf-42a0-97b2-93c1e19032a1","responseType":"FULL"}
```

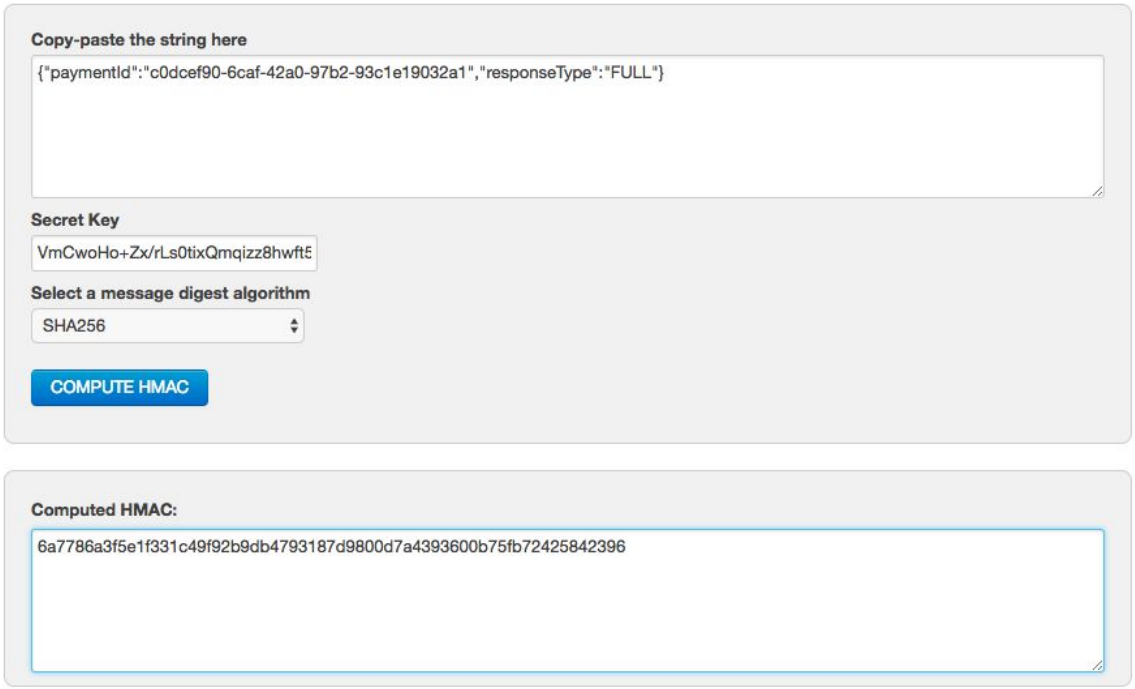
    - Checksum key

```
VmCwoHo+Zx/rLs0tixQmqizz8hwft5JR3MDkiXBqq2g=
```

    - Generate checksum hash

```
[linux]$ echo -n
'{"paymentId":"c0dcef90-6caf-42a0-97b2-93c1e19032a1","responseType":"FULL"}
' | openssl dgst -sha256 -hmac
'VmCwoHo+Zx/rLs0tixQmqizz8hwft5JR3MDkiXBqq2g='
(stdin) = 6a7786a3f5e1f331c49f92b9db4793187d9800d7a4393600b75fb72425842396
```

- Here is an example using online tool



II. ENUM definitions

1. Payment type

- topup
- conversion
- ccard
- move

2. Payment status

- pending
- processing
- success
- canceled
- failed
- expired
- confirmed
- pending_move_fund

III. Payment Endpoint API

1. Get Payment Detail

- Description: Get user's payment detail
- Url: https://xapi.kryptono.exchange/k/papi/v1/enterprise/user-payment-detail
- Method: POST
- Header:

Key	Value
X-Requested-With	XMLHttpRequest
Content-Type	application/json
X-KRP-APIKEY	{{server-key}}
Checksum	{{checksum hash}}

- Parameter:

Name	Type	Mandatory	Description
paymentId	STRING	YES	Payment's id
responseType	STRING	NO	Response type can be "SIMPLE" or "FULL". Default is "SIMPLE"

- Any Payment with type "conversion" or "ccard" may contain a "enterprisePaymentDetail".
- If responseType is "FULL", enterprisePaymentDetail will be included in response body.
- if responseType is "SIMPLE" or not, enterprisePaymentDetail will be excluded in response body.

- Response body with "responseType = SIMPLE"


```
{
    "id": "5b61f616892faf688fffd2b0",
    "paymentId": "c0dcef90-6caf-42a0-97b2-93c1e19032a1",
    "userIdentityId": "userIDHere",
    "enterprisePaymentId": "d870511f-d7ea-480a-9e07-71a939e52025",
    "status": "success",
    "createdAt": 1533146646231,
    "amountTo": "70.92198582",
    "amountFrom": "10",
    "enterpriseFee": "0",
    "enterpriseFeeCurrency": "USD",
    "fromCurrency": "USD",
    "toCurrency": "GTO",
```

```
    "type": "ccard",
    "rate": "0.141",
    "rateAt": 1533146646231,
    "squarePaymentReceipt": {
        "process_fee": 0,
        "idempotency_key": "75d2601e-a018-4996-8241-dd520c33b08a",
        "square_transaction_id":
"ciwHkViIyMoUjg6hAxEISRUHI39vjBbU3CJRB7tXQ7OOUbbKb9SoUIhO",
        "card_type": "VISA",
        "error_code": null
    },
    "enterprisePaymentDetail": null
}
```


- Response body with “responseType = FULL”


```
{
    "id": "5b61f616892faf688fffd2b0",
    "paymentId": "c0dcef90-6caf-42a0-97b2-93c1e19032a1",
    "userIdentityId": "userIDHere",
    "enterprisePaymentId": "d870511f-d7ea-480a-9e07-71a939e52025",
    "status": "success",
    "createdAt": 1533146646231,
    "amountTo": "70.92198582",
    "amountFrom": "10",
    "enterpriseFee": "0",
    "enterpriseFeeCurrency": "USD",
    "fromCurrency": "USD",
    "toCurrency": "GTO",
    "type": "ccard",
    "rate": "0.141",
    "rateAt": 1533146646231,
    "squarePaymentReceipt": {
        "process_fee": 0,
        "idempotency_key": "75d2601e-a018-4996-8241-dd520c33b08a",
        "square_transaction_id":
"ciwHkViIyMoUjg6hAxEISRUHI39vjBbU3CJRB7tXQ7OOUbbKb9SoUIhO",
        "card_type": "VISA",
        "error_code": null
    },
    "enterprisePaymentDetail": {
        "id": "5b6bd10b892faf13d07adec8",
        "paymentId": "d870511f-d7ea-480a-9e07-71a939e52025",
        "userPaymentId": "c0dcef90-6caf-42a0-97b2-93c1e19032a1",
        "status": "success",
        "createdAt": 1533792523461,
```

```
        "amount": "70.92198582",
        "fee": "0",
        "currencyCode": "GTO"
    }
}
```

- Error Response:

```
{
    "error" : "400001",
    "error_description" : "Error Description"
}
```

- Error Detail:

Http Status Code	Error Code	Error Description
400	400501	Invalid api key
400	400503	Invalid checksum
404	404501	Enterprise account not found
404	404502	Receipt not found
406	406501	Key side is not acceptable
406	406505	Not allowed to get receipt detail
500	500000	Error while processing request